UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/567,948	02/10/2006	Alan Walter Stiemens	0231	4652

31665 7590 05/26/2010

PATENT DEPARTMENT
ROVI CORPORATION
2830 DE LA CRUZ BOULEVARD
SANTA CLARA, CA 95050

EXAMINER	
CHOWDHURY, ZIAUL A.	
ART UNIT	PAPER NUMBER
2192	

MAIL DATE	DELIVERY MODE
05/26/2010	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary	Application No.	Applicant(s)	
	10/567,948	STIEMENS ET AL.	
	Examiner	Art Unit	
	ZIAUL CHOWDHURY	2192	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☐ Responsive to communication(s) filed on *22 February 2010*.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *67-69,72 and 78-100* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *67-69, 72,78-100* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____.
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

Detailed Action

Status of Claims

1. Applicant's amended claims dated February 22nd, 2010 responding to the January 4th, 2010 Office Action provided in the rejection of claims 67-105.

2. Claims 67-69, 72, and 78-100 have been amended.

3. Claims 67-69, 72, and 78-100 are pending in the application, of which claims 67, 78, 87 and 97 are in independent form and these claims have been fully considered by the examiner.

Remarks

4. Applicant's arguments with respect to claims rejection have been considered, but are moot in view of the new grounds of rejection. See Granger et al. (US Patent Publication No 6,643,775 –art being made of record).

Claim Rejections - 35 USC § 102

> (e) the invention was described in (1) an application for patent, published under section 122(b), by another filed in the United States before the invention by the applicant for patent or (2) a patent granted on an application for patent by another filed in the United States before the invention by the applicant for patent, except that an international application filed under the treaty defined in section 351(a) shall have the effects for purposes of this subsection of an application filed in the United States only if the international application designated the United States and was published under Article 21(2) of such treaty in the English language.

5. Claims 67- 69, 72, 78, 80, 82-83, 86-89, 95-100 are rejected under 35 U.S.C. 102(e) as being anticipated by Granger et al. (US Patent Publication No 6,643,775 –art being made of record).

Per claim 67:

Granger discloses:

A method of producing obfuscated object code, the method comprising: *substituting an assignment of a variable in source code with a class template defining a plurality of functions of the variable* (Col 20:31-35 –The cross-compiler 160 also preferably generates tables that store input variables, temporary variables, constants, and other entities that are used by the obfuscated code, and imbeds these tables within the obfuscated source code file 162; Col 20:23-28 –The process of translating individual operations into larger numbers of operations is governed by a set of obfuscation rules (template) that are stored in a rules library (template library) 180. For example, the mapping library 180 may contain a rule that specifies that the operation C=A+B is to be converted into the sequence C=2A, B=2B, C=(C+B)/2), *each of the plurality of functions indexed by a key value and associated with a series of operations resulting in the assignment of the variable in a manner obfuscating such assignment* (Col 20:16-20 –The cross-compiler 160 preferably generates the obfuscated C code by converting pre-specified types of operations (additions, multiplications, logical operations, moves, etc.) into larger numbers of constituent operations); *and compiling the source code using a compiler to produce object code* (Col 19:51-55 –when the obfuscated source code is compiled, the resulting machine code is substantially the same in length and complexity as the machine code (object code) that would be produced from the non-obfuscated source code), *wherein the compiler inserts in the object code the series of operations associated with one of the plurality of functions of the variable identified by a key value provided as a parameter of the class template*

(Col 21:18-24 - FIG. 11 is a flow chart which illustrates one method that may be used by the de-optimizing cross compiler 160 or other obfuscation tool to generate the obfuscated code. In this example implementation, individual instructions are translated into longer, equivalent sequences of instructions (referred to herein as "obfuscation sequences") at the machine level ; Col 22:66-67, Col 23:1-3 - The initial function contains two machine-level instructions, not including the function declaration and return. In operation, the obfuscation rules are applied repeatedly to the function's instructions until the target obfuscation level is reached).

Per claim 68:

Granger discloses:

wherein the series of operations comprises a plurality of operations from a group of operations including of arithmetic operations and logical operations (Col 8:24-31 –the obfuscation tool may be configured to translate the mathematical operation C=A+B into either of the following equivalent sequences of operations: X=A/2, Y=B/2, C=2X+2Y; or C=2A, B=2B, C=(C+B)/2. Non-mathematical operations, such as logical operations and moves of data between registers and memory, can similarly be translated into less efficient, equivalent operations).

Per claim 69:

Granger discloses:

wherein the group of operations further includes operations which can reliably return a variable to its assigned value (Col 10:56-61 - The

decryption engine 30' implements a decryption algorithm which is the inverse of the algorithm used to encrypt the user data. Because the same seed value is used to generate the decryption key, the decryption key is the same as the encryption key, and the original block of user data is reproduced; Col 8:24-31 –the obfuscation tool may be configured to translate the mathematical operation C=A+B into either of the following equivalent sequences of operations: X=A/2, Y=B/2, C=2X+2Y; or C=2A, B=2B, C=(C+B)/2. Non-mathematical operations, such as logical operations and moves of data between registers and memory, can similarly be translated into less efficient, equivalent operations).

Per claim 72:

Granger discloses:

wherein the source code is written in the C++ programming language (Col 4:30-33 -The term "high-level code" (source code) refers to textual code written in a high-level programming language (such as C, C++, Pascal or Java) that provides a level of abstraction from the underlying machine language).

Per claim 78:

Granger discloses:

A method of producing storage media having a secured executable program thereon, the method comprising: of *obfuscation object code of a security program by substituting an assignment of a variable in source code of the security program with a class template defining a plurality of functions of the variable* (Col 20:31-35 –The cross-compiler 160 also

preferably generates tables that store input variables, temporary variables, constants, and other entities that are used by the obfuscated code, and imbeds these tables within the obfuscated source code file 162; Col 20:23-28 –The process of translating individual operations into larger numbers of operations is governed by a set of obfuscation rules (template) that are stored in a rules library (template library) 180. For example, the mapping library 180 may contain a rule that specifies that the operation C=A+B is to be converted into the sequence C=2A, B=2B, C=(C+B)/2), *each of the plurality of functions indexed by a key value and associated with a series of operations resulting in the assignment of the variable in a manner obfuscating such assignment* (Col 20:16-20 –The cross-compiler 160 preferably generates the obfuscated C code by converting pre-specified types of operations (additions, multiplications, logical operations, moves, etc.) into larger numbers of constituent operations), and *compiling the source code using a compiler to produce the object code* (Col 19:51-55 –when the obfuscated source code is compiled, the resulting machine code is substantially the same in length and complexity as the machine code (object code) that would be produced from the non-obfuscated source code), *wherein the compiler inserts in the object code the series of operations associated with one of the plurality of functions of the variable identified by a key value provided as a parameter of the class template* (Col 21:18-24 - FIG. 11 is a flow chart which illustrates one method that may be used by the de-optimizing cross compiler 160 or other obfuscation tool to generate the obfuscated code. In this example implementation, individual instructions are translated into longer, equivalent sequences of instructions (referred to herein as "obfuscation sequences") at the machine level ; Col 22:66-67, Col 23:1-3 - The initial function contains two machine-level instructions, not including the function declaration and return. In operation, the obfuscation rules are applied repeatedly to the function's instructions

until the target obfuscation level is reached); *securing an executable program by associating the executable program with the security program which is arranged to control access to the executable program* (Col 19:47-51 –Code obfuscators that are used for this purpose (exposing high-level source code to outsiders) operate by performing such tasks as replacing names of variables with arbitrarily-assigned character sequences, and merging multiple functions into larger blocks of code; Col 26:58-67, Col 27:1-5 -On a computer-readable medium, an application that is protected from unauthorized use, the application generated by performing at least the following steps: providing a code module which implements at least one function of a use-restriction scheme, the use-restriction scheme using an electronic security device (ESD) to prevent an unauthorized use of the application; processing at least a portion of said code module with an obfuscation tool to controllably increase a quantity of code used to implement the function of the use-restriction scheme, the obfuscation tool generating obfuscated code; and imbedding the obfuscated code within the application; wherein the obfuscated code inhibits the generation of a non-use-restricted version of the application); *and applying the secured executable program to a storage media* (Col 26:58-67, Col 27:1-5 -On a computer-readable medium, an application that is protected from unauthorized use, the application generated by performing at least the following steps: providing a code module which implements at least one function of a use-restriction scheme, the use-restriction scheme using an electronic security device (ESD) to prevent an unauthorized use of the application; processing at least a portion of said code module with an obfuscation tool to controllably increase a quantity of code used to implement the function of the use-restriction scheme, the obfuscation tool generating obfuscated code; and imbedding the obfuscated code within the application; wherein the obfuscated code inhibits the generation of a non-use-restricted version of the application).

Per claim 80:

Granger discloses:

further comprising moving blocks of the executable program out of the executable program and relocating the blocks in the security program (Col 3:18-24 -involves the intertwining of copy-protection and non-copy-protection functions within a single block of obfuscated code or pseudocode. A non-copy-protection function that is necessary to the proper operation of the application is preferably used for this purpose, so that attempts to remove the block of code from the application will render the application inoperative).

Per claim 82:

Granger discloses:

wherein the source code of the security program involves stored arrays and templates and utilizes pointers to navigate the arrays and templates (Col 18:27-34 - the SPEC decrypts and processes an ECODE data block (generated by the EASM) that is stored in the computer's local memory. The location of the ECODE block is passed to the SPEC by the calling function via a pointer. The ECODE data block needs to reside in a modifiable partition if writing to local memory is required. Arguments can optionally be passed with the pointer, and have the effect of pre-loading SPEC registers. Arguments can represent pointers or numerical operands).

Per claim 83:

Granger discloses:

wherein the source cede of the security program is written in the C++ programming language (Col 4:30-33 -The term "high-level code" (source code) refers to textual code written in a high-level programming language (such as C, C++, Pascal or Java) that provides a level of abstraction from the underlying machine language).

Per claim 86:

Granger discloses:

wherein the storage media onto which the secured executable program is applied is a memory unit associated with at least one computer (Col 1:20-25 –The present invention relates to methods for preventing the unauthorized distribution and use of computer programs. More particularly, the present invention relates to methods for impairing the ability of software pirates to remove or disable the executable copy protection code, or other security code, within a computer program).

Per claim 87:

Granger discloses:

A storage media having an executable program and a security program thereon (Col 26:58-67, Col 27:1-5 -On a computer-readable medium, an application that is protected from unauthorized use, the application generated by performing at least the following steps: providing a code module which implements at least one function of a use-restriction

scheme, the use-restriction scheme using an electronic security device (ESD) to prevent an unauthorized use of the application; processing at least a portion of said code module with an obfuscation tool to controllably increase a quantity of code used to implement the function of the use-restriction scheme, the obfuscation tool generating obfuscated code; and imbedding the obfuscated code within the application; wherein the obfuscated code inhibits the generation of a non-use-restricted version of the application), *wherein the security program controls access to the executable program), the security program is in object code* (Col 19:23-33 –the Obfuscation Layer is preferably combined with the Encryption Layer by implementing one or more of the following Encryption Layer components (all shown in FIGS. 3A and 3B), or portions thereof, in obfuscated code: the condenser 44, the expander 42, the ESD simulator 32' (including the table decryptor 46'), the encryption engine 30, and the decryption engine 30'. As with the Pseudocode Layer, the level of copy protection provided by the Obfuscation Layer can be increased by intermingling copy-protection and non-copy protection functions within a single, callable block of obfuscated code), and *the object code of the security program has been obfuscated by substituting an assignment of a variable in source code of the security program with a class template defining a plurality of functions of the variable* (Col 20:31-35 –The cross-compiler 160 also preferably generates tables that store input variables, temporary variables, constants, and other entities that are used by the obfuscated code, and imbeds these tables within the obfuscated source code file 162; Col 20:23-28 –The process of translating individual operations into larger numbers of operations is governed by a set of obfuscation rules (template) that are stored in a rules library (template library) 180. For example, the mapping library 180 may contain a rule that specifies that the operation C=A+B is to be converted into the sequence C=2A, B=2B, C=(C+B)/2), *each of the plurality of*

functions indexed by a key value and associated with a series of operations resulting in the assignment of the variable in a manner obfuscating such assignment (Col 20:16-20 –The cross-compiler 160 preferably generates the obfuscated C code by converting pre-specified types of operations (additions, multiplications, logical operations, moves, etc.) into larger numbers of constituent operations), and *compiling the source code using a compiler to produce the object code* (Col 19:51-55 – when the obfuscated source code is compiled, the resulting machine code is substantially the same in length and complexity as the machine code (object code) that would be produced from the non-obfuscated source code), w*herein the compiler inserted in the object code the series of operations associated with one of the plurality of functions of the variable identified by a key value provided as a parameter of the class template* (Col 21:18-24 - FIG. 11 is a flow chart which illustrates one method that may be used by the de-optimizing cross compiler 160 or other obfuscation tool to generate the obfuscated code. In this example implementation, individual instructions are translated into longer, equivalent sequences of instructions (referred to herein as "obfuscation sequences") at the machine level ; Col 22:66-67, Col 23:1-3 - The initial function contains two machine-level instructions, not including the function declaration and return. In operation, the obfuscation rules are applied repeatedly to the function's instructions until the target obfuscation level is reached).

Per claim 88:

Granger discloses:

wherein the series of operations comprises a plurality of operations from a group of operations including arithmetic operations and logical operations

(Col 8:24-31 –the obfuscation tool may be configured to translate the mathematical operation C=A+B into either of the following equivalent sequences of operations: X=A/2, Y=B/2, C=2X+2Y; or C=2A, B=2B, C=(C+B)/2. Non-mathematical operations, such as logical operations and moves of data between registers and memory, can similarly be translated into less efficient, equivalent operations).

Per claim 89:

Granger discloses:

wherein the group of operations further includes operations which can reliably return a variable to its assigned value (Col 10:56-61 - The decryption engine 30' implements a decryption algorithm which is the inverse of the algorithm used to encrypt the user data. Because the same seed value is used to generate the decryption key, the decryption key is the same as the encryption key, and the original block of user data is reproduced; Col 8:24-31 –the obfuscation tool may be configured to translate the mathematical operation C=A+B into either of the following equivalent sequences of operations: X=A/2, Y=B/2, C=2X+2Y; or C=2A, B=2B, C=(C+B)/2. Non-mathematical operations, such as logical operations and moves of data between registers and memory, can similarly be translated into less efficient, equivalent operations).

Per claim 95:

Granger discloses:

wherein the storage media is a memory associated with at least one computer (Col 1:20-25 –The present invention relates to methods for

preventing the unauthorized distribution and use of computer programs. More particularly, the present invention relates to methods for impairing the ability of software pirates to remove or disable the executable copy protection code, or other security code, within a computer program).

Per claim 97:

Granger discloses:

A method implemented by a first processor for providing a program for secure execution on a second processor, the method comprising: *receiving a key value* (Col 2:38-41 -The first method involves using values generated by a conventional ESD (Electronic Security Device) to encrypt and/or decrypt user data (such as a file) that is generated and used by the application); using the key value to retrieve information of a template from a library of templates indexed by key values (Col 2:58-65 –the second method involves using pseudocode to implement some or all of the application's copy protection or other use-authorization functions. The pseudocode for a given function is generated (preferably in encrypted form) from actual code using a special development tool, and is then imbedded within the application together with a corresponding pseudocode interpreter; Col 20:23-28 –The process of translating individual operations into larger numbers of operations is governed by a set of obfuscation rules (template) that are stored in a rules library (template library) 180. For example, the mapping library 180 may contain a rule that specifies that the operation C=A+B is to be converted into the sequence C=2A, B=2B, C=(C+B)/2), wherein each of the templates in the library is associated with a different encrypted set of instructions that result from compiling source code of the program and associated with a different emulator program configured to translate

corresponding of the different encrypted set of instructions into a set of obfuscated native instructions (Col 2:61-67,Col 3:1-2 - The pseudocode for a given function is generated (preferably in encrypted form) from actual code using a special development tool, and is then imbedded within the application together with a corresponding pseudocode interpreter. The interpreter fetches, decrypts and executes the pseudocode when the function is called. Because no disassemblers or other development tools exist for analyzing the pseudocode, the task of analyzing the copy protection functions becomes significantly more complex):

compiling the source code to generate an encrypted set of instructions associated with the template (Col 19:51-55 –when the obfuscated source code is compiled, the resulting machine code is substantially the same in length and complexity as the machine code (object code) that would be produced from the non-obfuscated source code; Col 19:23-26 -the Obfuscation Layer is preferably combined with the Encryption Layer by implementing one or more of the following Encryption Layer components (all shown in FIGS. 3A and 3B), or portions thereof, in obfuscated code), wherein the encrypted set of instructions is not directly executable by the second processor (Col 2:61-67,Col 3:1-2 - The pseudocode for a given function is generated (preferably in encrypted form) from actual code using a special development tool, and is then imbedded within the application together with a corresponding pseudocode interpreter. The interpreter fetches, decrypts and executes the pseudocode when the function is called. Because no disassemblers or other development tools exist for analyzing the pseudocode, the task of analyzing the copy protection functions becomes significantly more complex); and providing the encrypted set of instructions along with an emulator program associated with the template to the second processor (Col 2:61-65 –The pseudocode for a given function is generated (preferably in encrypted

form) from actual code using a special development tool (emulator), and is then imbedded within the application together with a corresponding pseudocode interpreter), wherein the emulator program and the set of obfuscated native instructions are directly executable by the second processor (Col 2:65-66 –The interpreter fetches, decrypts and executes the pseudocode when the function is called).

Per claim 98:

Granger discloses:

wherein the emulator program is configured to translate the encrypted set of instructions into the associated set of obfuscated native instructions without decrypting the encrypted set of instructions (Col 21:8-18 - the obfuscated C source code file 162 (and any other such files that may be generated by the same process) is compiled together with the other application source code files 174 (using a regular C compiler 176) to generate the application's machine code file(s) 178. During this process, the obfuscated C source code is translated into a proportional quantity of obfuscated machine code. Thus, the obfuscation at the C level translates into obfuscation at the machine level. The obfuscated functions are called during the execution of the application in the same manner as are non-obfuscated functions).

Per claim 99:

Granger discloses:

Further comprising providing an executable program along with the encrypted set of instructions and the emulator program to the second

processor (Col 2:61-66 -The pseudocode for a given function is generated (preferably in encrypted form) from actual code using a special development tool, and is then imbedded within the application together with a corresponding pseudocode interpreter. The interpreter fetches, decrypts and executes the pseudocode when the function is called), *wherein the encrypted set of instructions control access to the executable program so that the second processor is allowed to execute the executable program* (Col 21:8-18 - the obfuscated C source code file 162 (and any other such files that may be generated by the same process) is compiled together with the other application source code files 174 (using a regular C compiler 176) to generate the application's machine code file(s) 178. During this process, the obfuscated C source code is translated into a proportional quantity of obfuscated machine code. Thus, the obfuscation at the C level translates into obfuscation at the machine level. The obfuscated functions are called during the execution of the application in the same manner as are non-obfuscated functions).

Per claim 100:

Granger discloses:

wherein the emulator program translates the encrypted set of instructions into the obfuscated set of native instructions by translating at least one of the encrypted set of instructions into a string of native instructions to obfuscate the native instructions (Col 2:61-66 -The pseudocode for a given function is generated (preferably in encrypted form) from actual code using a special development tool, and is then imbedded within the application together with a corresponding pseudocode interpreter. The interpreter fetches, decrypts and executes the pseudocode when the function is called).

Claim Rejections - 35 USC § 103

6. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

7. Claims 79, 81, 84, 85, 90-94 are rejected under 35 U.S.C. 103(a) as being unpatentable over Granger et al. in view of Chen et al. (US Patent Application Publication No 2004/ 0003278 A1).

Per claim 79:

Granger does not disclose:

wherein the executable program and the security program are associated at object code level, the security program being arranged to encrypt the executable program;

However, Chen discloses

the executable program and the security program are associated at object code level (Paragraph 57; The compiler module 712 processes the source code following the OTL code substitution operation to generate object code that is combined with other executable modules by the linker module 713 to generate an executable processing module), the security program being arranged to encrypt the executable program (Paragraph 16; One aspect of the present invention is a system for providing secure and opaque type libraries to automatically provide secure variables

within a programming module (encrypt). The system includes an OTL selection module, an OTL substitution module, an OTL type library database, a compiler module; and a linker module to create an executable processing module).
It would have been obvious to one ordinary skill in the art at time the invention was made to combine Granger and Chen because that would further enhance Granger with the capability package the executable code with the security code together, and therefore, the object code can be protected against piracy while running the program or distributing (–*See* Chen Paragraph 2).

Per claim 81:

Granger does not disclose:

wherein the security program is arranged to require the running of an authentication program;

However, Chen discloses :
wherein the security program is arranged to require the running of an authentication program (Paragraph 40; It is known how (by carefully composing elementary arithmetic operations with random constants) to generate authentication tag generation mechanisms (reversible and non reversible) randomly and they may be used here to automatically choose encryption and authentication mechanisms). Thus, it would have been obvious to one ordinary skill in the art at time the invention was made to combine Granger's method with Chen's method because that would sufficiently enhance the security program by adding extra authentication program layer on the top of the security program which would protect obfuscated functions as well the return value due to the function calls (–*See* Chen Paragraph 35).

Per claim 84:

Granger does not disclose:

wherein the storage media onto which the secured executable program is applied is an optical disc;

However, Chen discloses:

wherein the storage media onto which the secured executable program is applied is an optical disc (Paragraph 44; A number of program modules may be stored on the hard disk, magnetic disk 216, optical disk 219, ROM 208 or RAM 210, including an operating system 226, one or more application programs 228, other program modules 230, and program data 232). It would have been obvious to one ordinary skill in the art to store executable program and security program together on an optical disc.

It would have been obvious to one ordinary skill in the art at time the invention was made to combine Granger with Chen because Chen's method which would give Granger's method the capability to save the secured executable codes in a storage medium such as an optical disc, therefore, it could be read by computer or any other electronic device which are capable of reading executable instruction, data structures, and other data including security programs (–*See* Chen Paragraph 42).

Per claim 85:

Granger does not disclose:

wherein the secured executable program is applied to the optical disc by laser beam encoding;

However, Chen discloses:

wherein the secured executable program is applied to the optical disc by laser beam encoding (Paragraph 42; The computer system 200 further includes a hard disk drive 212 for reading from and writing to a hard disk, a magnetic disk drive 214 for reading from or writing to a removable magnetic disk 216, and an optical disk drive 218 for reading from or writing to (done by laser beam encoding) a removable optical disk 219 such as a CD ROM, DVD, or other optical media –emphasis added).

It would have been obvious to one ordinary skill in the art and industries to encode the executable program and security program together on an optical disc using laser beam).

It would have been obvious to one ordinary skill in the art at time the invention was made to combine Granger with Chen because Chen's method which would give Granger's method the capability to save the secured executable codes in a storage medium such as an optical disc, therefore, it could be read by computer or any other electronic device which are capable of reading executable instruction, data structures, and other data including security programs (–*See* Chen Paragraph 42).

Per claim 90:

Granger does not explicitly disclose:

wherein the executable program and the security program are associated at object code level , and wherein the executable program is encrypted on the storage media and the associated security program enables decryption of the executable program.

However, Chen discloses

the executable program and the security program are associated at object code level (Paragraph 57; The compiler module 712 processes the source

code following the OTL code substitution operation to generate object code that is combined with other executable modules by the linker module 713 to generate an executable processing module), *and wherein the executable program is encrypted on the storage media and the associated security program enables decryption of the executable program* (Paragraph 16; One aspect of the present invention is a system for providing secure and opaque type libraries to automatically provide secure variables within a programming module (encrypt). The system includes an OTL selection module, an OTL substitution module, an OTL type library database, a compiler module; and a linker module to create an executable processing module).

It would have been obvious to one ordinary skill in the art at time the invention was made to combine Granger and Chen because that would further enhance Granger with the capability package the executable code with the security code together, and therefore, the object code can be protected against piracy while running the program or distributing (–*See* Chen Paragraph 2).

Per claim 91:

Granger does not disclose:

wherein blocks from the executable program have been relocated within the security program;

Chen discloses :

wherein blocks from the executable program have been relocated within the security program (Paragraph 57; The compiler module 712 processes the source code following the OTL code substitution operation to generate object code that is combined with other executable modules by the linker module 713 to generate an executable processing module).

It would have been obvious to one ordinary skill in the art at time of the invention was made to combine Granger's method with Chen's method because that would facilitate Granger's method to embed the executable program in the security program.

Per claim 92:

Granger does not disclose:

wherein the security program is arranged to require the running of an authentication program;

However, Chen discloses :
wherein the security program is arranged to require the running of an authentication program (Paragraph 40; It is known how (by carefully composing elementary arithmetic operations with random constants) to generate authentication tag generation mechanisms (reversible and non reversible) randomly and they may be used here to automatically choose encryption and authentication mechanisms). Thus, it would have been obvious to one ordinary skill in the art at time the invention was made to combine Granger's method with Chen's method because that would sufficiently enhance the security program by adding extra authentication program layer on the top of the security program which would protect obfuscated functions as well the return value due to the function calls (– *See* Chen Paragraph 35).

Per claim 93:

Granger does not disclose:

wherein the storage media onto which the secured executable program is applied is an optical disc.

Chen discloses :

wherein the storage media onto which the secured executable program is applied is an optical disc (Paragraph 44; A number of program modules may be stored on the hard disk, magnetic disk 216, optical disk 219, ROM 208 or RAM 210, including an operating system 226, one or more application programs 228, other program modules 230, and program data 232). It would have been obvious to one ordinary skill in the art to store executable program and security program together on an optical disc.

It would have been obvious to one ordinary skill in the art at time the invention was made to combine Granger with Chen because Chen's method which would give Granger's method the capability to save the secured executable codes in a storage medium such as an optical disc, therefore, it could be read by computer or any other electronic device which are capable of reading executable instruction, data structures, and other data including security programs (–*See* Chen Paragraph 42).

Per claim 94:

Granger does not disclose:

wherein the optical disc is one of: a CD, a CD-ROM, and a DVD;

However, Chen discloses:

wherein the optical disc is one of: a CD, a CD-ROM, and a DVD (Paragraph 42; The computer system 200 further includes a hard disk drive 212 for reading from and writing to a hard disk, a magnetic disk drive 214 for reading from or writing to a removable magnetic disk 216,

and an optical disk drive 218 for reading from or writing to (done by laser beam encoding) a removable optical disk 219 such as a CD ROM, DVD, or other optical media –emphasis added).

It would have been obvious to one ordinary skill in the art and industries to encode the executable program and security program together on an optical disc using laser beam.

It would have been obvious to one ordinary skill in the art at time the invention was made to combine Granger with Chen because Chen's method which would give Granger's method the capability to save the secured executable codes in a storage medium such as an optical disc, therefore, it could be read by computer or any other electronic device which are capable of reading executable instruction, data structures, and other data including security programs (–*See* Chen Paragraph 42).

Per claim 96:

Granger does not disclose:

wherein the executable program is one or more of: a games program; a video program; an audio program; and other software;

However, Chen discloses :

wherein the executable program is one or more of: a games program; a video program; an audio program; and other software (Paragraph 44; Examples of other input devices may include a microphone, joystick, game pad, satellite dish, and scanner. For hand-held devices and tablet PC devices, electronic pen input devices may also be used. These and other input devices are often connected to the processing unit 202 through a serial port interface 240 that is coupled to the system bus 206. Nevertheless, these input devices also may be connected by other

interfaces, such as a parallel port, game port, or a universal serial bus (USB)).

It would have been obvious to one ordinary skill in the art at time the invention was made to combine Granger with Chen because Chen's method would further provide an enhance capability to Granger's method for utilize the security programs to protect the actual game, audio, and video programs to be protected against unauthorized distribution or copying (–*See* Chen Paragraph 5).

Conclusion

8. Applicant's amendment necessitated the new ground(s) of rejection presented in this Office action. Accordingly, **THIS ACTION IS MADE FINAL**. See MPEP § 706.07(a). Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).
A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the date of this final action.

9. Any inquiry concerning this communication or earlier communications from the examiner should be directed to ZIAUL CHOWDHURY whose telephone number is (571)270-7750. The examiner

can normally be reached on Monday Thru Friday, 7:30AM To 9:00PM, Alternet Friday, Eastern Time.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, TUAN Q. DAM can be reached on 571-272-3695. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

/ZIAUL CHOWDHURY/
Examiner, Art Unit 2192

/Lewis A. Bullock, Jr./
Supervisory Patent Examiner, Art Unit 2193